Appendix:

VHDL Toplevel code

Binarytree.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.package_sjs.all;
entity binary_tree is
	generic (
		level : integer;
		dp_width : integer;--:= 4;
		blockram_width : integer ; --:= 16;
	col_len : integer; -- := 4;
	data_width :integer -- := 64
	);
	port (
		--inputs
		val_a : in aval;
		col : in acol;
		zeropad : in asig;
		write_enable : in std_logic;
		clock : in std_logic;
		ready : in std_logic;
		exception_in : in std_logic;
		round : in std_logic;
		row_id_x : in integer;
		--outputs
		out1 : out std_logic_vector(data_width-1 downto 0);
		root_done : out std_logic;
		exception_out: out std_logic;
		data_x_T : out std_logic_vector(data_width-1 downto 0)
	);
end entity binary_tree;
architecture binary_tree_arch of binary_tree is
signal s_exc : asig;
signal s_done: asig;
signal s_out1: aval;
begin
	isupp: component supply
		generic map (
			dp_width ,
			blockram_width,
			col_len,
			data_width
		)
		port map (
			val_a => val_a,
			col => col,
			zeropad => zeropad,
			write_enable => write_enable ,
			clock => clock,
			ready => ready,
			exception_in => exception_in,
```

```
                round => round,

                row_id_x => row_id_x,
                out1 => s_out1,
                exception_out => s_exc,
                done => s_done,
        data_x_T => data_x_T
        );
    itreee: component tree
        generic map (
                level,
                data_width,
                dp_width
        )
        port map(
                val_in => s_out1,
                ready => s_done,
                exception_in => s_exc,
                round => round,
                clock => clock,
                --outputs
                out1 => out1 ,
                root_done => root_done,
                exception_out => exception_out
        );
end architecture binary_tree_arch;
```

Control.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--use IEEE.numeric_std.all;
use work.package_sjs.all;
entity control is
	generic (
		rows : integer;
		dp_width : integer; --:= 4
		data_width : integer; --:= 32
		col_len : integer --:= 16
	);
	port (
		clock : in std_logic;
		root_done : in std_logic;
		reset : in std_logic;
		supply_over : in std_logic;
		data_x_T_1 : in std_logic_vector(data_width-1 downto 0);
		read_addr_x_T_1 : out integer; -- read addr for bram x(t+1)
		write_data : out std_logic := '0';
		row_id_in : in integer;
		red_done : in std_logic ; -- reduction unit done
		data_diff : in std_logic_vector(data_width-2 downto 0); -- |x(t+1)-x(t)|
		done_diff : in std_logic; -- difference started to come
	    row_id_conv : in integer;  -- used to stop the convergence
			-- checking for the particular iteration
		val_a  : in aval; -- array of std_logic_vector(data_width-1 downto 0);
		col    : in acol; -- array of std_logic_vector(col_len-1 downto 0);
		zeropad : out asig; -- array of std_logic
		write_enable : out std_logic := '1';
		done : out std_logic ;
		-----------------
		--port for bram B
		-----------------
		col_b : out std_logic_vector(col_len-1 downto 0);
		col_b_read : out std_logic_vector(col_len-1 downto 0);
		write_b : out std_logic := '1';
		--------------------
		-- port for divider
		--------------------
		val_div : out std_logic_vector(data_width-1 downto 0);
		div_ready : out std_logic;
		write_addr : out std_logic_vector(col_len-1 downto 0);
		--------------------
		--------------------------
		-- Port for the Bram len--
		--------------------------
		len : in std_logic_vector(col_len-1 downto 0);
		write_addr_len : out std_logic_vector(col_len-1 downto 0);
```

```
                read_addr_len : out std_logic_vector(col_len-1 downto 0);
                we_len : out std_logic := '1';
                data_len : out std_logic_vector(col_len-1 downto 0);
                data_out_len : in std_logic_vector(col_len-1 downto 0);
                -------------------------
                tag : out integer;
                row_id : out integer
        );
end entity control;
architecture control_arch of control is
        signal next_iter : std_logic;
        --- Shift value = log(dp_width)
        constant shift_val : unsigned := "10";
        --for supply
        type supply_state is (init, supply, write_res, flush, the_end);
        signal s_state : supply_state ;
        ------------
        --for tag generation
        type tag_state is (tag_genr, tag_flush, tag_end);
        signal s_tagstate : tag_state;
--for convergence
        type conv_state is (conv_wait, conv_check, conv_flush, conv_end);
        signal s_convstate : conv_state;
begin
        pctrl : process (clock, reset)

        variable iter_no : integer := 0;
        variable vaddr_len : integer := 1;
--      variable counter_clone : integer ;
--      variable vtag : integer := 1;
--      variable vtag : integer := 0;
        variable vlenu : unsigned (col_len-1 downto 0);
        variable vrow_id : integer := 1 ; -- changed from 0
        variable vxT1 : integer := 1;
        variable vshift_val : integer;
begin
                if (reset = '1') then
                        s_state <= init;
                elsif rising_edge(clock) then
                        case s_state is
                        when init       =>  - Read the length vector and store the tag in the BRAM
                                                --      counter_clone := CONV_INTEGER(len);
                                                --      vtag := 1;
                                                --      while (counter_clone > dp_width) loop
                                                --              vtag := vtag + 1;
                                                --      counter_clone := counter_clone - dp_width;
                                                --      end loop;
                vlenu := shr(UNSIGNED(len), shift_val);
                                        --      if (len(0) /= '0' or len(1) /= '0') then
                                        --              vlenu := vlenu + 1;
                                                        --      end if;
                                        vshift_val := CONV_INTEGER(shift_val);
                                        for i in 0 to vshift_val-1 loop
```

```
                    if(len(i) /= '0') then
                vlenu := vlenu + 1;
                    exit;
                                        end if;
                                    end loop;

                                    -- supply the address and the data BRAM
                                    write_addr_len <=
CONV_STD_LOGIC_VECTOR(vaddr_len, col_len);
                                    data_len <= STD_LOGIC_VECTOR(vlenu);
                                    if (vaddr_len = rows) then
                                        s_state <= supply;
                                        write_enable <= '0';
                                    --      we_len <= '0';
                                    end if;
                                    vaddr_len := vaddr_len + 1;
                            when supply     =>
                                    for i in 0 to dp_width-1 loop
                    if (vrow_id = CONV_INTEGER(col((i+1)*col_len-1 downto i*col_len))) then
                    zeropad(i) <= '1';  -- used to zeropad the multiplier when i=j
val_div <= val_a((i+1)*data_width-1 downto i*data_width);     -- supply the value to the divider
            -- note that divider is getting the value

            -- one cycle earlier than the binary tree
        div_ready <= '1' ; -- divider is ready
        write_addr <= CONV_STD_LOGIC_VECTOR(vrow_id, col_len);
        vrow_id := vrow_id + 1;     -- increment when Aii case matches
                                        else
                                            zeropad(i) <= '0';
                                        end if ;

                                    end loop;

                                        -- added to remove the
bug....when consequetive coloumns are present in case of state supply

                                    for i in 0 to dp_width-1 loop
                                        if (vrow_id = CONV_INTEGER(col(
(i+1)*col_len-1 downto i*col_len) ) ) then

                                            vrow_id := vrow_id + 1;
                                            exit;
                                        end if;
                                    end loop;

                                    done <= '1';

                                    if(supply_over = '1') then
                                        s_state <= write_res; --next state
condition

                                        iter_no := iter_no + 1;
                                    end if;
```

```
                                        when write_res   =>              ------------------------------------
                                                                        --Write the X values to the output--
                                                                        ------------------------------------

                                                                        read_addr_x_T_1 <= vxT1;

                                                                        if (vxT1 > 2) then              -- we skip this
twice so that the data_x_T_1 becomes available
                                                                                write_data <= '1';
                                                                                if (vxT1 = vrow_id + 2) then -- and
thus added 2 to the vrow_id
                                                                                        s_state <= flush;
                                                                                end if;
                                                                        end if;
                                                                vxT1 := vxT1 +1 ; -- next row
                                                when flush      =>              ------------------------------------
                                                                        --Start flushing the pipeline
                                                                        ------------------------------------
                                                                        done <= '0';
                                                                        div_ready <= '0';

                                                                        if (done_diff = '0') then  -- acts as a
representative for FLUSH OVER
                                                                                if (next_iter = '0') then  --
permanently wait if the convegence is satisfied
                                                                                        s_state <= the_end;
                                                                                else
                                                                                        s_state <= init;
                                                                                        write_enable <= '1';
                                                                                        write_data <= '0';
                                                                                        -- Initialise variables for
the next iteration
                                                                                        vaddr_len := 1;
                                                                                --      vtag := 1;
                                                                                        --vtag := 0;
                                                                                        --vtag_count := 1;
                                                                                        vrow_id := 1;
                                                                                        vxT1 := 1;
                                                                                        ----- check new variables
added
                                                                                end if;
                                                                        end if;

                                                when the_end    =>              s_state <= the_end;         -- wait forever
                                end case;
                        end if;
                end process pctrl;
                ptag_gen : process (clock, reset)
                        variable vtag : integer := 0;
                        variable vrow_id : integer := -1;
                        variable vlen_addr : integer := 1;
                        variable skip : integer := 1;
begin
```

```
		if (reset = '1') then
			s_tagstate <= tag_genr;
		elsif rising_edge(clock) then
			case s_tagstate is
				when tag_genr	=>		if (root_done = '1') then
								if(vtag = 0) then
									read_addr_len <=
CONV_STD_LOGIC_VECTOR(vlen_addr, col_len);
									we_len <= '0';
									vlen_addr := vlen_addr +
1;
									if (skip = 0) then
										vtag :=
CONV_INTEGER(data_out_len);
									end if;
									vrow_id := vrow_id + 1;
								end if;
								if (skip = 0) then
									tag <= vtag;
									row_id <= vrow_id;
									vtag := vtag-1;
								end if;
								skip := 0; -- logic to skip the
assignement state twice
							end if;

							if (vrow_id = rows and vtag = 0) then
								s_tagstate <= tag_flush;
							end if;
				when tag_flush	=>	if (done_diff = '0') then  -- acts as a representative
for FLUSH OVER
							if (next_iter = '0') then  -- permanently wait
if the convegence is satisfied
								s_tagstate <= tag_end;
							else
								s_tagstate <= tag_genr;
								-- Initialise variables for the next
iteration
								vtag := 0;
								vrow_id := -1;
								vlen_addr := 1;
								skip := 1;
							end if;
						end if;
				when tag_end	=>	s_tagstate <= tag_end; -- wait forever
			end case;
		end if;
	end process ptag_gen;
	--==============END TAG GENERATION==================---
	--------------------
	-- STORING B VALUE--
	--------------------
	pbstore: process (clock)
```

```
		variable vbcol : integer := 1;

	begin
		if (falling_edge(clock)) then
			if (vbcol <= rows) then
				col_b <= CONV_STD_LOGIC_VECTOR(vbcol, col_len);
				vbcol := vbcol + 1;
			end if;
		end if;
	end process pbstore;
	pbread	: process (clock)
		begin
		-- see whether reduction circuit starts outputing
		if (rising_edge(clock)) then
			if (red_done = '1') then
				write_b <= '0';
				col_b_read <= CONV_STD_LOGIC_VECTOR (row_id_in, col_len);
			end if;
		end if;
	end process pbread;
	--=============END STORING B VALUE=============---
	-----------------------------------------
	--CHECK THE CONVERGENCE CONDITION--
	pchk_conv : process (clock, reset)
	constant TOL : std_logic_vector := x"3727C5AC"; -- tolerance = 1e-10
	begin
		if (reset = '1') then
			s_convstate <= conv_wait;
		elsif rising_edge(clock) then
			case s_convstate is
				when conv_wait		=>	next_iter <= '0';
								if (done_diff = '1') then
								--if (rising_edge(done_diff)) then
									s_convstate <= conv_check;
								end if;
				when conv_check		=>	if (data_diff > TOL) then
									next_iter <= '1';
								end if;
								if(row_id_conv = rows) then
									s_convstate <= conv_flush;
								end if;
				when conv_flush		=>	if (done_diff = '0') then  -- acts as a
representative for FLUSH OVER
									if (next_iter = '0') then  --
permanently wait if the convegence is satisfied
										s_convstate <= conv_end;
									else
										s_convstate <= conv_wait;
									end if;
								end if;
				when conv_end		=>	s_convstate <= conv_end; -- wait forever
			end case;
		end if;
```

```
        end process pchk_conv;
end architecture control_arch;
```

MATLAB Top level Code:

MATLABtreealgo.m

```
clear
j=sqrt(-1);
baseMVA=100;
baseKV=10;
baseZ=baseKV^2/baseMVA;

getdata
linedatas
getybus
getfamily
newtonraphson
Iterations
J
getlineflow
originalloss=LOSS
originalfamily=family
PARENT=family(:,1);
CHILD=family(:,2);
LINE=family(:,3);
losses=[];
counter=0;

branches=numel(CHILD);
nodes(1)=0;
for xyz=1:branches
    for abc=1:(branches+1)
        if abc==CHILD(xyz)
            nodes(abc)=PARENT(xyz);
        end
    end
end
nodes;
treeplot(nodes)
pause

for dd=1:numel(LINE)
    TEST=[];
    for ee=1:KK
        if bnum(ee)==LINE(dd)
            testbus=CHILD(dd)
            breakerdata(ee,8)=0;
        end
    end    %takes line number from LINE and makes the child of that line the
test bus, and turns of that line

    for ff=1:KK
        if b1(ff)==testbus & breakerdata(ff,8)==0
            TEST=[TEST;breakerdata(ff,:)];
        elseif b2(ff)==testbus & breakerdata(ff,8)==0
            TEST=[TEST;breakerdata(ff,:)];
        end
```

```
    end    %creates a matrix with the breakerdata of each 'off' line
connected to the test bus

   TEST;
    for hh=1:(numel(TEST)/8)
            for gg=1:KK
            if TEST(hh,7)==bnum(gg)
                breakerdata(gg,8)=1;    %find each possible parent line and
turn them on one at a time
                linedatas
                getybus
                newtonraphson2
                getlineflow2
                counter=counter+1;
                losses=[losses; bnum(gg), LOSS];
                breakerdata(gg,8)=0;
                lossmag=losses(:,2);
                lossline=losses(:,1);   %calculate losses with each possible
parent line turned on, one at a time
            end
        end
    end
    losses
    for mm=1:length(lossline)
        if lossmag(mm)==min(lossmag)        %find which of the possible
parent lines results in the least losses
            oline=lossline(mm);
            for nn=1:KK
                if bnum(nn)==oline
                   breakerdata(nn,8)=1;     %turn on the line that results in
the least losses
                   chosenline=nn;
                end
            end
        end
    end
    losses=[];
    chosenline
end                 %go on to next testbus


linedatas
getybus
getfamily
newtonraphson
getlineflow             %final lineflow calculation with the optimal
configuration set
optimalloss=LOSS;

originalfamily
family

originalloss
optimalloss
counter

branches=numel(CHILD);
```

```
nodes(1)=0;
for xyz=1:branches
    for abc=1:(branches+1)
        if abc==CHILD(xyz)
            nodes(abc)=PARENT(xyz);
        end
    end
end
nodes;
treeplot(nodes)
```

newtonraphson.m

```
s=0; vc=0; Vm=0; delta=0; yload=0; deltad=0;
code=[];Vm=[]; delta=[]; Pd=[]; Qd=[]; Pg=[]; Qg=[]; Qmin=[]; Qmax=[];
Pk=[]; P=[]; Qk=[]; Q=[]; S=[]; V=[];

for k=1:buses
    n=busdata(k,1);
    code(n)=busdata(k,2);
    Vm(n)=busdata(k,3);
    delta(n)=busdata(k,4);
    Pd(n)=busdata(k,5);Qd(n)=busdata(k,6);
    Pg(n)=busdata(k,7);Qg(n)=busdata(k,8);
    Qmin(n)=busdata(k,9);Qmax(n)=busdata(k,10);
    Qshunt(n)=busdata(k, 11);
    if Vm(n)<=0                 %
        Vm(n)=1.0;
        V(n)=1+j*0;
        else delta(n)=pi/180*delta(n);
            V(n)=Vm(n)*(cos(delta(n))+j*sin(delta(n)));
            P(n)=(Pg(n)-Pd(n))/baseMVA;
            Q(n)=(Qg(n)-Qd(n)+ Qshunt(n))/baseMVA;
            S(n)=P(n)+j*Q(n);
    end
end

for k=1:buses
    if code(k)==1
        s = s+1;
    else, end
    if code(k)==2
        vc=vc+1;
    else, end
    nvc(k) = vc;
    ns(k) = s;
end

Ym=abs(Ybus);
t=angle(Ybus);
m=2*buses-vc-2*s;                   %figure out how big J will be
accuracy=.000001;
maxiter=80;
maxerror=.00001;
converge=1;
iter=0;

%Starting Iterations
clear A J DX DC


while maxerror>=accuracy & iter<=maxiter        % Test for max. power
mismatch
    for h=1:m
        for k=1:m
```

```
            J(h,k)=0;      %Initializing Jacobian matrix
        end
    end

    iter = iter+1;
    for n=1:buses
        nn=n-ns(n);
        lm=buses+n-nvc(n)-ns(n)-s;          %inner dimensions

        J11=0; J1=0;
    J22=0; J2=0;
    J33=0; J3=0;
    J44=0; J4=0;
        for h=1:lines
            if na(h)==n | nb(h)==n
                if na(h)==n
                    l=nb(h);
                end
                if nb(h)==n
                    l=na(h);
                end

                J11=J11+Vm(n)*Vm(l)*Ym(n,l)*sin(t(n,l)-delta(n)+delta(l));
                J33=J33+Vm(n)*Vm(l)*Ym(n,l)*cos(t(n,l)-delta(n)+delta(l));

                if code(n)~=1
                    J22=J22+ Vm(l)*Ym(n,l)*cos(t(n,l)- delta(n) + delta(l));
                    J44=J44+ Vm(l)*Ym(n,l)*sin(t(n,l)- delta(n) + delta(l));
                else, end


                if code(n)~=1 & code(l)~=1
                    lk=buses+l-nvc(l)-ns(l)-s;
                    ll=l-ns(l);                  %inner dimensions

                        J(nn,ll)=-Vm(n)*Vm(l)*Ym(n,l)*sin(t(n,l)-
delta(n)+delta(l));
                        % off diagonal elements of J1
                        if code(l)==0
                            J(nn,lk)=Vm(n)*Ym(n,l)*cos(t(n,l)-
delta(n)+delta(l));
                            % off diagonal elements of J2
                        end
                        if code(n)==0
                            J(lm,ll)=-Vm(n)*Vm(l)*Ym(n,l)*cos(t(n,l)-
delta(n)+delta(l));
                             % off diagonal elements of J3
                        end
                        if code(n)==0 & code(l)==0
                            J(lm,lk)=-Vm(n)*Ym(n,l)*sin(t(n,l)-
delta(n)+delta(l));
                             % off diagonal elements of  J4
                        end

                else end
            else end
        end
```

```
    Pk=Vm(n)^2*Ym(n,n)*cos(t(n,n))+J33;
    Qk=-Vm(n)^2*Ym(n,n)*sin(t(n,n))-J11;



    if code(n)~=1
         J(nn,nn)=J11;                                    %diagonal elements of
J1
         DC(nn)=P(n)-Pk;
     end
     if code(n)==0
         J(nn,lm)=2*Vm(n)*Ym(n,n)*cos(t(n,n))+J22;        %diagonal elements of
J2
         J(lm,nn)=J33;                                    %diagonal elements of
J3
         J(lm,lm)=-2*Vm(n)*Ym(n,n)*sin(t(n,n))-J44;       %diagonal elements of
J4
         DC(lm)=Q(n)-Qk;
     end
   end



J;
DC;
DCd=DC';

%SEND INFO TO FPGA (WITH ITER SET TO 0)

%CSRconv

%SOLVING VIA MATLAB (WITH PROPER ITER AND ERROR SET)

DX=J\DC';
     for n=1:buses;
         nn=n-ns(n);
         lm=buses+n-nvc(n)-ns(n)-s;
         if code(n) ~= 1
             delta(n) = delta(n)+DX(nn);
         end
         if code(n) == 0
             Vm(n)=Vm(n)+DX(lm);
         end
     end
     maxerror=max(abs(DC));
         if iter == maxiter & maxerror > accuracy
         fprintf('\nSOLUTION DOES NOT CONVERGE AFTER ')
         fprintf('%g', iter), fprintf(' ITERATIONS\n\n')
         fprintf('PRESS ENTER TO PRINT RESULTS \n')
         converge = 0; pause,
         else end

end

J;
```

```
Iterations=iter;

V = Vm.*cos(delta)+j*Vm.*sin(delta);
deltad=180/pi*delta;
j=sqrt(-1);
k=0;
for n=1:buses
     if code(n)==1
     k=k+1;
     S(n)=P(n)+j*Q(n);
     Pg(n)=P(n)*baseMVA+Pd(n);
     Qg(n)=Q(n)*baseMVA+Qd(n)-Qshunt(n);
     Pgg(k)=Pg(n);
     Qgg(k)=Qg(n);
     elseif  code(n)==2
     k=k+1;
     S(n)=P(n)+j*Q(n);
     Qg(n)=Q(n)*baseMVA+Qd(n)-Qshunt(n);
     Pgg(k)=Pg(n);
     Qgg(k)=Qg(n);
     end
     yload(n)=(Pd(n)-j*Qd(n)+j*Qshunt(n))/(baseMVA*Vm(n)^2);
end
busdata(:,3)=Vm'; busdata(:,4)=deltad';
Pgt=sum(Pg);  Qgt=sum(Qg); Pdt=sum(Pd); Qdt=sum(Qd); Qshuntt=sum(Qshunt);
```